Idea of Flow-LLM

September 23, 2025

1 Motivation

Vocabulary: V, with size $= |V|$, tokenized as $V = \{1, 2, \cdots, |V|\}$.

Denote one sample consisting of L token ids as

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_L \end{bmatrix}, \quad \text{where each token } x_l \in V.$$

Let the joint multinomial probability of the L tokens on the vocabulary be

$$\Pi = \begin{bmatrix} \pi_1 \\ \vdots \\ \pi_L \end{bmatrix} = \begin{bmatrix} \pi_1^1 & \cdots & \pi_1^{|V|} \\ \vdots & & \vdots \\ \pi_L^1 & \cdots & \pi_L^{|V|} \end{bmatrix}_{(L, |V|)}, \quad \text{where each } \pi_l \in \mathcal{S}_{|V|-1}.$$

$\mathcal{S}_{|V|-1}$ is the $(|V|-1)$-probability simplex:

$$\sum_{v=1}^{|V|} \pi_l^v = 1 \text{ and } \pi_l^v \geq 0 \text{ for } v = 1, ..., |V|$$

So Π lies in a multi-dimensional probability simplex. And our goal is to approximate $p_{\text{true}}(\Pi)$ and generate Π on the multi-dimensional simplex.

In other words, **instead of doing generative modeling directly on the tokens x (a discrete problem), we do modeling on their probabilities Π over the vocabulary V (a continuous problem).** After generating samples of Π, we can pick tokens using argmax or sampling from multinomial distribution.

Assume the following:

- Let the current observed samples of $p_{\text{true}}(\Pi)$ be the one-hot encoding of the input sample tokens. For example, the following is one sample of $p_{\text{true}}(\Pi)$:

$$\begin{bmatrix} x_1 \\ \vdots \\ x_L \end{bmatrix} = \begin{bmatrix} \text{one-hot encoding of } x_1 \\ \vdots \\ \text{one-hot encoding of } x_L \end{bmatrix}_{(L, |V|)}$$

- Assume prior distribution $p_{\text{init}}(\Pi)$ to be independent Dirichlet distribution:

$$p_{\text{init}}(\Pi) = \prod_{l=1}^{L} p_{\text{init}}(\pi_l) = \prod_{l=1}^{L} \text{Dirichlet}(\mathbb{1}_{|V|}/|V|).$$

fine-tuning, domain-adaption, (char?) shot learning
SRI

2 Training target of flow-matching

The goal of flow matching is to find vector field such that we can make the "probability path flow to the true unknown probability distribution", i.e.,

- marginally

$$
\begin{aligned}
p_0(\Pi) &= p_{\text{init}}(\Pi) = \text{some simple distribution}, \\
p_1(\Pi) &= p_{\text{true}}(\Pi) = \text{true unknown distribution of } \Pi.
\end{aligned}
$$

- conditionally:

$$
p_0(\Pi|z) = p_{\text{init}}(\Pi), \quad p_1(\Pi|z) = \delta_z.
$$

One of the most popular choice that satisfies the above goal is linear path

$$
\Pi_t = (1-t)\Pi_0 + tz, \quad t \in [0,1]
$$

where $\Pi_0 \sim p_{\text{init}}$, $z \sim p_{\text{true}}$. The correponding conditional vector field is

$$
u_t(\Pi|z) = \frac{z - \Pi}{1-t}, \quad t \in [0,1).
$$

Loss function:

$$
\begin{aligned}
\mathcal{L}(\theta) &= \mathbb{E}_{t\sim\text{Unif}(0,1), z\sim p_{\text{true}}, \Pi\sim p_t(\Pi|z)} \|u_t^\theta(\Pi) - u_t^{\text{target}}(\Pi|z)\|^2 \\
&= \mathbb{E}_{t\sim\text{Unif}(0,1), z\sim p_{\text{true}}, \Pi\sim p_t(\Pi|z)} \|u_t^\theta(\Pi) - (z-\Pi)/(1-t)\|^2 \\
&= \mathbb{E}_{t\sim\text{Unif}(0,1), z\sim p_{\text{true}}, \Pi_0\sim p_{\text{init}}} \|u_t^\theta(tz + (1-t)\Pi_0) - (z-\Pi_0)\|^2.
\end{aligned}
$$

Algorithm 1: Flow matching training procedure

for *each mini-batch of data* **do**
  Sample $z \sim p_{\text{true}}$ given the mini-batch;
  Sample $t \sim \text{Uniform}(0,1)$;
  Sample $\Pi_0 \sim \prod_{l=1}^{L} \text{Dirichlet}(\mathbb{1}_{|V|}/|V|)$;
  Set $\Pi = (1-t)\Pi_0 + tz$;
  Compute loss $\mathcal{L}(\theta) = \|u_t^\theta(\Pi) - (z-\Pi_0)\|^2$;
  Update the model parameter θ by gradient descent on $\mathcal{L}(\theta)$.
end
output: Learned vector field u_t^θ

3 Sampling

Aftering learning the vector field u_t^θ, we can generate samples of Π by Euler method:

Algorithm 2: Sampling from a flow model with Euler method

Set $t = 0$;
Set step size $h = 1/n$;
Draw an initial sample $\Pi_0 \sim p_{\text{init}}$;
for $i = 1, ..., n-1$ **do**
  $\Pi_{t+h} = \Pi_t + h u_t^\theta(\Pi_t)$;
  Update $t \leftarrow t + h$
end
output: Π_1

4 Architecture

```
prob_embedding = self.prob_emb(xt) # (batch_size, n_tokens, emb_size)
position_embedding = self.position_emb(self.position_ids) # (n_tokens, emb_size)
time_embedding = self.time_emb(t)  # (batch_size, emb_size)

x = prob_embedding + position_embedding # (batch_size, n_tokens, emb_size)
x = self.dropout(x)

for block in self.trf_blocks: # Several transformer blocks with multi-head attention
    x = block(x, time_embedding)

x = self.final_norm(x) # (batch_size, n_tokens, emb_dim)
vec_field = self.out_linear(x) # (batch_size, n_tokens, vocab_size)
```